Chapter 10

Experimental Efficiency Analysis

In Chapters 6 and 7, we introduced three methods to evaluate approXQL queries: The *direct* method operates on lists that store a subset of all data-tree nodes with a given value. Both the *schema* method and the *hybrid* method use the path tree (schema) of the data tree to find the best k transformed queries, which are then executed against the data tree. The *schema* method creates a new second-level query for each combination of matching schema paths, whereas the *hybrid* method merges second-level queries with similar substructures.

Given the three query-evaluation methods, our first objective in this chapter is to analyze the dependencies of the query-evaluation times on several query and data parameters. The second objective is to compare the query-evaluation methods in order to determine which approach is superior for several specific parameter settings. Based on these findings, a future version of the approXQL query engine can be enabled to choose the appropriate evaluation method after analyzing the indexed collection and the query being evaluated.

The chapter is organized as follows: In the first section, we formulate five hypotheses, which we will verify in the rest of this chapter. In Section 10.2, we describe the configuration of our test system, the auxiliary software employed for query and data generation, and the characteristics of the query patterns and XML document collections used for the tests. In Section 10.3, we describe the test series that have been carried out. Section 10.4 provides a detailed analysis of the results obtained during the experiments. Finally, in Section 10.5, we summarize the main results.

10.1 Hypotheses

From the variety of topics that are worth investigating, we select those that focus on characteristics we think are crucial for the fast similarity search: (1) the ability to quickly retrieve the best n results for different types of queries; (2) the correlation between the query-evaluation times and the number of permitted value changes per query selector; the influence of the schema size on the query-evaluation times assuming that the number of distinct element names (3) grows according to the schema size or (4) remains constant; and (5) the dependencies of the query-evaluation times on the selectivities of the query terms. In the following, we formulate a hypothesis for each of the topics.

1. **Number of requested results:** The evaluation time of a query using the *schema* method or the *hybrid* method increases with the number of requested results. The evaluation time using *direct* method remains constant.

2. **Number of value changes:** The evaluation time of a query increases as the number of value changes per selector increases.

3. **Schema size, varying number of names:** Using a constant collection size, the evaluation time of a query decreases as the schema size increases, provided that the number of distinct element and attribute names increases with the schema size.

4. **Schema size, constant number of names:** Using a constant collection size and a constant number of distinct element and attribute names, the evaluation time of a query using the *schema* method or the *hybrid* method increases as the schema size increases. The evaluation time using the *direct* method remains constant.

5. **Term selectivity:** The evaluation time of a query using the *direct* method increases as the term selectivities decrease. The evaluation time of a query using the *schema* method or the *hybrid* method decreases as the selectivities decrease.

Note that we do not investigate the dependencies of the evaluation times on the number of deletions, permutations, and insertions here. The number of deletions is restricted by the number of query selectors; the number of permutations is restricted by the number of inner selectors that have ancestor-descendant relationships. Each deletion or permutation encoded in a query-execution plan increases the evaluation times by a constant factor. Node insertions have no influence on the evaluation times, because the insertion costs are calculated in constant time.

10.2 Experimental Setup

In this section, we review the characteristics of the computer used for our test series, the configuration of important parameters of the approXQL query engine, and the auxiliary software used to create random document collections and queries. We also describe the characteristics of the document collections and queries used in our experiments.

10.2.1 System Configuration

We carried out all test series on an Intel Pentium 4 computer with 1.25 GHz and 512 MB memory, running Linux 2.4.18 as its operating system. The approXQL query engine is implemented on top of the Berkeley Database version 3.3.11 [BER02], using B-trees to implement the indexes described in Chapter 8. We configured the database kernel with a page size of 64 KB, in order to store long index postings in a single page. The cache size of the database kernel was set to 10 MB.

Recall from Section 6.6.2 that the approXQL query engine uses dynamic programming to implement the evaluation of query-execution plans. Depending on the buffer size for storing intermediate results, the system can avoid the recomputation of shared subplans. For the *schema* method and the *hybrid* method, the system stores the results of the subqueries of second-level queries in order to avoid multiple evaluations of identical subqueries belonging to subsequent second-level queries. We used a cache size of 100 MB, which is sufficient to avoid the recomputation of shared subplans/subqueries even for complex queries.

A crucial parameter for both the *schema* method and the *hybrid* method is k, the number of second-level queries required to create n results. As discussed in Section 7.4, the choice of k depends on the characteristics of the query and the data, and is very hard to predict. In all our tests, we calculated k by the formula,

$$k = \max(200, 3 \cdot n),$$

which proved to be a good compromise between the additional time needed to create unused second-level queries (if fewer queries were needed to find n results), and the additional time needed to increase the number of second-level queries (if the best k queries retrieved less than n results). We used the factor 2 to increase k after each cycle.

To ensure the controlled behavior of the system even in extreme cases, we limited the evaluation time of an approXQL query to 5 minutes. Furthermore, for the *schema* method and the *hybrid* method, we canceled the evaluation of a query if k reached a limit of 100,000, or if the set of second-level queries consumed more than 100 MB of memory.

10.2.2 Software used for Experimental Setup

In this section, we describe the two tools used for preparing our experiments: the Niagara-project XML data generator [ANZ01] and the approXQL query generator.

The Niagara-Project XML Data Generator

The Niagara-project XML data generator produces XML files according to parameters provided in a configuration file. A DTD is neither required nor supported. To generate a collection of XML documents, two sets of parameters have to be adjusted: The first set controls the characteristics of the path tree, e.g., its depth, the minimum and maximum fanout of the nodes at each level, the fraction of internal path-tree nodes with recursion in their tag names, and the fraction of internal path-tree nodes with repetition in their tag names. The second set controls the characteristics of the documents created according to the path tree, e.g., the number of elements per document, the total number of distinct words (terms), the Zipf value (skew) of the term-frequency distribution, and the maximum number of term occurrences per element. The variety of parameters allow to produce XML document collections with very different characteristics. However, the configuration is not always trivial because many parameters depend on each other.

The approXQL Query Generator

The choice of test queries is crucial for analyzing the efficiency of a query processor. We decided not to use a fixed set of hand-coded queries, because the test queries might have unwanted dependencies resulting from the subjective choice of the selection values.

In our experiments, all queries have been produced by a simple generator: The generator expects a query pattern, which determines the structure of the query. A query pattern consists of name templates, term templates, and operators. The query generator produces approXQL queries by filling in the templates with concrete names and terms. A predefined percentage of names and terms do not appear in the collection of documents; all other names and terms are selected randomly from the indexes.

For each produced query, the generator also creates a cost file. The file for a query selector contains the deletion costs of the selector and a predefined number of alternative values, together with their value-change costs. All costs are randomly chosen from a predefined interval. The values are selected randomly from the indexes. All deletion costs are less than infinite, which means that all query selectors may be deleted.

The random selection of terms (and names) is the default behavior of the query generator. It additionally supports the selection of the most frequent terms (maxfreq mode) and the least frequent term (minfreq mode) as follows: If the generator is in maxfreq mode, then it creates a list of all terms L occurring in the collection and sorts the list by decreasing frequency. If the query pattern has t term templates and v alternative values per selector, then the generator selects the t first terms from L, and assigns them randomly to the term templates. The alternative values are randomly chosen from the interval $L[t, t + t \cdot v - 1]$. In the minfreq mode, the generator works similarly, but assumes L to be sorted by increasing frequency.

If the generator works in maxfreq mode, then the terms of the created queries are the most frequent ones found in the collection. Therefore, the queries have *minimum term selectivities*. Similarly, if the generator works in minfreq mode, then the queries have *maximum term selectivities*.

10.2.3 XML Collections

There is still a shortage of large test collections of XML documents. The collections publicly available are either far too small to allow experiments with expressive results (like the Sigmod record [Mer99]), or they have a regular structure with a very small schema (like the religious-works collection [Bos99]). For our experiments, we chose the DBLP collection, which consists of XML documents containing bibliographic data about conferences, journals, and books published in the field of database research. The documents in the DBLP collection are quit regularly structured, but the collection is large enough to allow a meaningful analysis of the query-evaluation times.

From all of the synthetic document collections used in our experiments, we selected five for the systematic analysis presented in this thesis. These collections allow us to analyze and compare the evaluation times of a query with respect to (i) different schema sizes, (ii) different numbers of element names using constant schema sizes, and (iii) different term distributions.

Table 10.1 on the following page introduces the names of the collections and summarizes their characteristics. The row “names” lists the total number of distinct element and attribute

collection	*dblp*	*100x100*	*1000x100*	*1000x1000*	*10000x100*	*10000x10000*
documents	282,151	100				
names	41	100	100	1,000	100	10,000
elements	3,217,918	1,000,000				
terms	471,357	100,000				
words	9,729,638	10,000,000				
Zipf value	≈ 1	1				
schema nodes	148	100	1,000		10,000	
nesting levels	5	6	7		9	

Table 10.1: Characteristics of the XML document collections used in the experiments.

names occurring in the collection; the row "elements" lists the total number of elements and attributes. (The synthetic collections do not contain attributes, because the data generator cannot produce them.) Further, the row "terms" lists the number of distinct words in the collections; the row "words" lists the total number of words (term occurrences). The Zipf value determines the skew of a term frequency distribution according to Zipf's law $P_i \sim 1/i^z$ [Zip49]. The function calculates the number of occurrences P_i of a term in a collection, where z is the Zipf parameter and i is the position of the term in the frequency-sorted list of all terms occurring in the collection. The Zipf value 1 describes a term distribution similar to the one found in typical English texts [MNF58]. Synonymous names for the row "nesting levels" are "depth of the data tree" and "depth of the path tree", where the depth is counted by including the nodes of type *data*.

The *1000x100* and *10000x100* collections required an additional postprocessing of the XML files created by the data generator: To yield a schema size of 1000 or 10000 nodes with a constant number of 100 distinct names, we ran a simple program that reduced the number of different names, but kept the relative frequencies of the names.

10.2.4 Query Patterns

In our experiments, we initialized the query generator with a wide range of query patterns and measured the evaluation time of the generated approXQL queries. For the analysis presented in this chapter, we exemplarily choose four patterns that represent a "simple path query", a "small Boolean query", a "large Boolean query", and a "structure-only" query, respectively. The patterns are shown in Table 10.2 on the next page. The names of the first three patterns reflect the number of hierarchical levels (first digit) and the number of leaf selectors (last

pattern name	pattern definition
4x1	`name[name[name[term]]]`
3x3	`name[name[term and (term or term)]]`
5x7	`name[name[name[term and term and (term or term)] or name[name[term and term]]] and name]`
struct	`name[name[name and (name or name)]]`

Table 10.2: Query patterns used in the experiments.

digit) of the pattern. The *struct* pattern has the same shape as the *3x3* pattern, except that only structural selectors exist. We configured the query generator to use the interval [1..10] for the assignment of deletion and value-change costs. All insertion costs were set to 1, and no permutations were defined.

10.3 Performed Experiments

To allow the verification of our hypotheses with respect to all combinations of collections, query patterns, and evaluation methods, we defined a hierarchy of test series (see Figure 10.1).

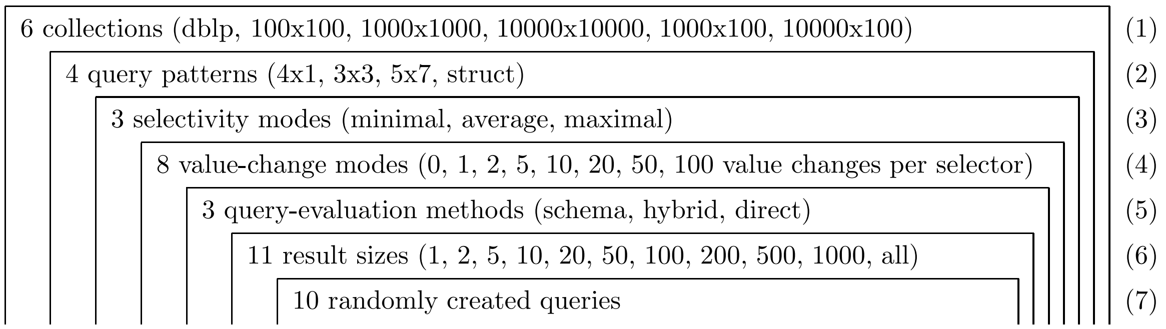

6 collections (dblp, 100x100, 1000x1000, 10000x10000, 1000x100, 10000x100) (1)
4 query patterns (4x1, 3x3, 5x7, struct) (2)
3 selectivity modes (minimal, average, maximal) (3)
8 value-change modes (0, 1, 2, 5, 10, 20, 50, 100 value changes per selector) (4)
3 query-evaluation methods (schema, hybrid, direct) (5)
11 result sizes (1, 2, 5, 10, 20, 50, 100, 200, 500, 1000, all) (6)
10 randomly created queries (7)

Figure 10.1: Hierarchy of test series.

With each of the six XML document collections (Line 1 in Figure 10.1) we used all four query patterns (Line 2). Given a pair of collection and query pattern, we carried out three test series (Line 3): In the first series, we configured the query generator to run in maxfreq mode (see Section 10.2.2) in order to produce queries with minimum selectivities. In the second series, the generator produced queries with average term selectivities. Finally, in the third series, the generator was configured to run in minfreq mode. For each combination of collection, query pattern, and selectivity, we carried out eight test series, where a different number of value changes per query selector was defined for each series (Line 4). At this stage, we ran

name	description
direct method	The direct query-evaluation method (proposed in Chapter 6).
schema method	The (non-optimized) schema-driven query-evaluation method (proposed in Chapter 7).
hybrid method	The schema-driven query-evaluation method that merges second-level queries with similar substructures (proposed in Chapter 7).
generation time	The time needed to generate the appropriate number of second-level queries for the retrieval of the requested number of results. Not defined for the *direct* method.
evaluation time	Total time needed to evaluate a query. For the *schema* method and the *hybrid* method, this is the total time needed to generate and evaluate second-level queries.
average selectivity	The query terms are randomly selected from the list of all terms occurring in the collection.
minimum selectivity	The query terms are randomly selected from the list the most frequent collection terms.
maximum selectivity	The query terms are randomly selected from the list the least frequent collection terms.

Table 10.3: Summary of names used for the query-evaluation methods, for time measurement, and for the choice of query terms.

the query generator in order to derive 10 random queries from the current query pattern, and to create the cost file for the generated queries. The 10 queries were then evaluated using each of the three methods (Line 5). If either the *schema* method or the *hybrid* method was used, we carried out 11 subsequent series. Each of these series used a different value for n, the number of requested results (Line 6). Because the *direct* method does not support the estimation of the best n results, we only carried out a single series. Each measured time value is the mean of the evaluation times of the 10 queries (Line 7).

All measured times are real times that include disc-access times and computation times. To obtain the times we used the Unix system call `gettimeofday()`. Table 10.3 summarizes the names used for the query-evaluation methods, for time measurement, and for the choice of query terms.

10.4 Results of the Experiments

In this section, we present the results of our experiments and confirm or dismiss the five hypotheses formulated in Section 10.1. Due to the large number of results, we only present a selection of representative diagrams in each subsection.

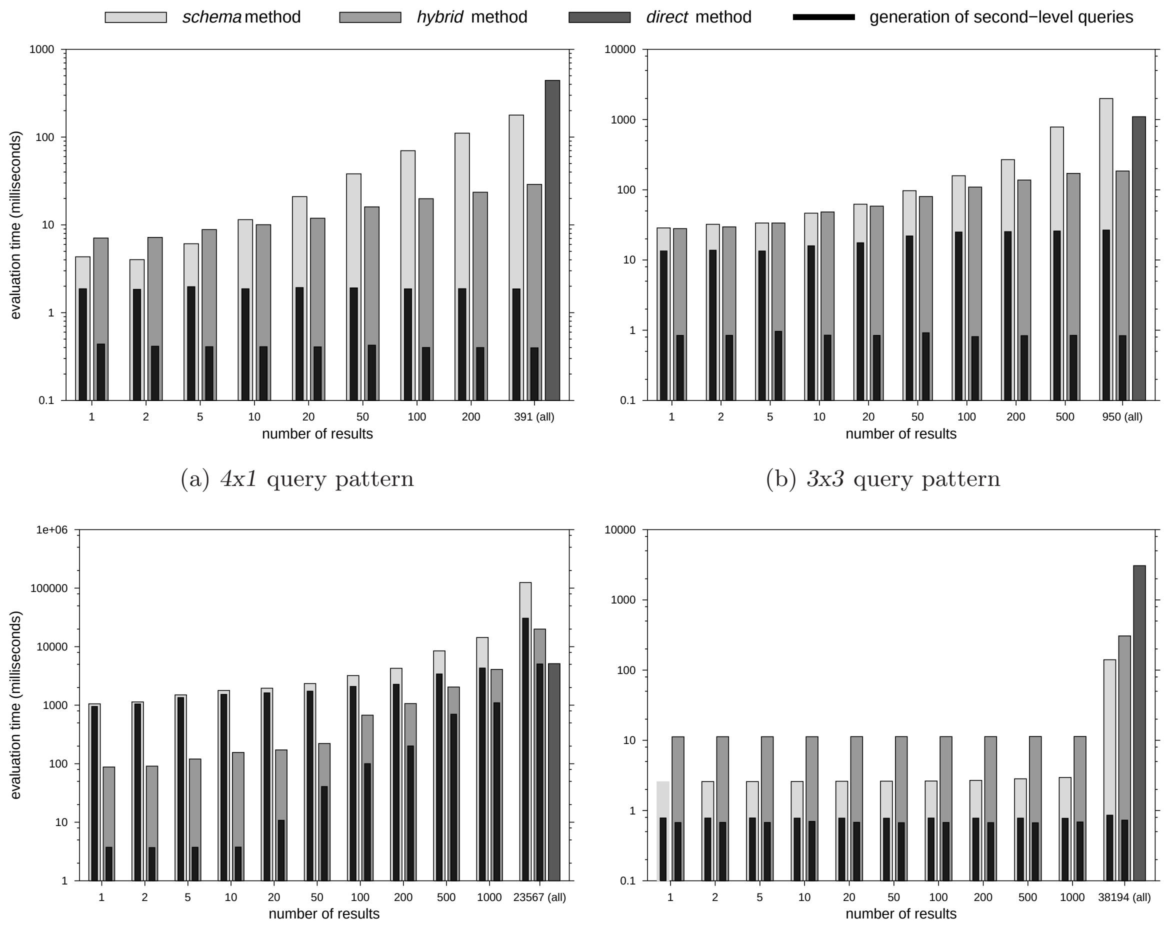
10.4.1 Hypothesis 1 (Number of Requested Results)

Hypothesis 1 states that the evaluation time of a query using the *schema* method or the *hybrid* method increases as the number of requested results increases, and that the evaluation time of a query using the *direct* method is independent from the number of results. In fact, the second part of the hypothesis is necessarily true because the *direct* method must compute all results to retrieve a prefix of the best ones.

We present the dependencies of the evaluation times on the number of results with respect to the collections *100x100* and *dblp* using a fixed number of 10 value changes per query selector. All evaluation methods show a similar time behavior if other collections and other number of value changes are used.

Figure 10.2 on the following page shows the evaluation times for the four query patterns with respect to the *100x100* collection. The x-axis of each diagram denotes n, the number of requested results for the queries; the y-axis shows the evaluation times. Note that the y-axes have logarithmic scales. For each number of results, the diagrams show two bars representing the evaluation time of the *schema* method and the *hybrid* method, respectively. At the x-position representing the retrieval of all existing results, each diagram additionally displays a bar that shows the evaluation time using the *direct* method. The bars drawn for the *schema* method and the *hybrid* method include narrow black bars, which show the total time needed to create a sufficient number of second-level queries, which in turn select the requested number of results. Each bar in the diagrams represents the mean evaluation time of 10 queries randomly generated for the same pattern. Note that a time value, measured using either the *schema* method or the *hybrid* method, may include the times of several evaluation cycles needed to increase the number of second-level queries.

Figure 10.2(a) shows the evaluation times of the *4x1* query. The *schema* method and the *hybrid* method both outperform the *direct* method in all cases — even if all results are requested. In fact, our experiments have shown that the *schema* method and the *hybrid* method are still faster than the *direct* method if we do not allow value changes, and even if we forbid



(a) *4x1* query pattern

(b) *3x3* query pattern

(c) *5x7* query pattern

(d) *struct* query pattern

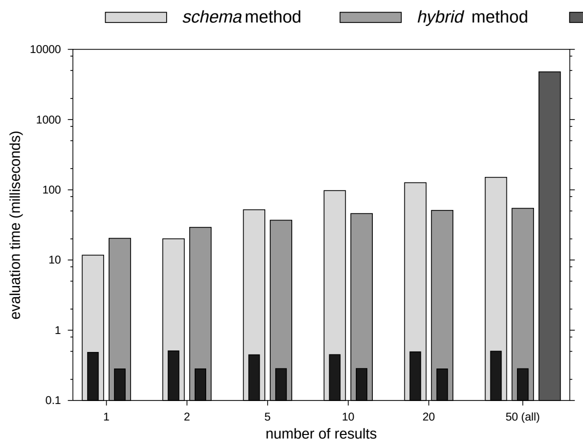
Figure 10.2: Query-evaluation times for the *100x100* collection depending on the number of requested results. Fixed parameter: 10 value changes per query selector.

the deletion of nodes. There are two reasons for this behavior: First, all second-level queries created have at least one embedding in the data tree (each second-level query is a type-value path). Second, during the evaluation of a second-level query, only the *node instances* (see Definition 7.3 on page 111) of the query nodes are considered. Those are all nodes with a certain type and value, and with a certain position in a type-value path. In contrast, the algorithm for the *direct* method inspects *all* matches of a query node in the data tree. The higher selectivities of query nodes in the *schema* method result in shorter lists, and therefore in faster list operations. The *schema* method outperforms the *hybrid* method if very few results are requested. However, the evaluation time of the *hybrid* method increases far less than the evaluation time of the *schema* method as the number of requested results increases.

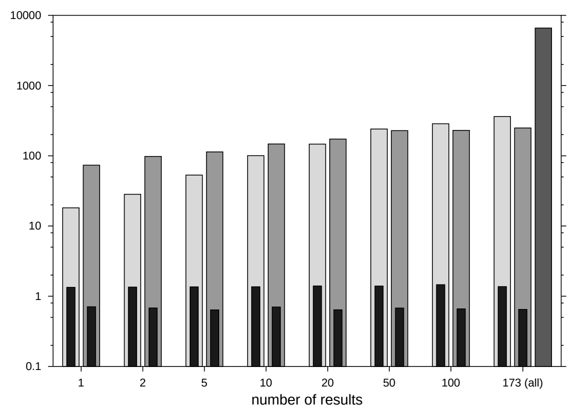
The evaluation times of the *3x3* query pattern are shown in Figure 10.2(b). The diagram shows that the evaluation times of the *3x3* query pattern using the *schema* method and the *hybrid* method are higher than the evaluation times of the *4x1* query pattern (note the different scales of both y-axes). The reason is that some created queries may find no results, and a larger k must be chosen. Figure 10.2(b) also shows an important difference between the *schema* method and the *hybrid* method: The *schema* method is slower in all cases and its evaluation times again rise faster than those of the *hybrid* method. In particular, the query-creation times (represented by the black bars) consume most of the total evaluation times. This indicates that many "useless" second-level queries have been created and evaluated (each with low evaluation time) until enough results were found. It follows that the *hybrid* method has a better ratio of the number of second-level queries created to the number of results per query than the *schema* method.

Figure 10.2(c) supports this observation: The *schema* method is slower than the *hybrid* one, and almost all of the processing time is spent on creating second-level queries. However, also the *hybrid* method becomes slow if many results are requested. This indicates that the merging of second-level queries does not fully avoid the creation of "useless" queries. Further experiments have shown that this does not only hold for the *5x7* query, but is a general effect of an increasing number of "`and`" operators. For queries with many "`and`" operators, the *direct* method is superior to the two schema-driven methods.

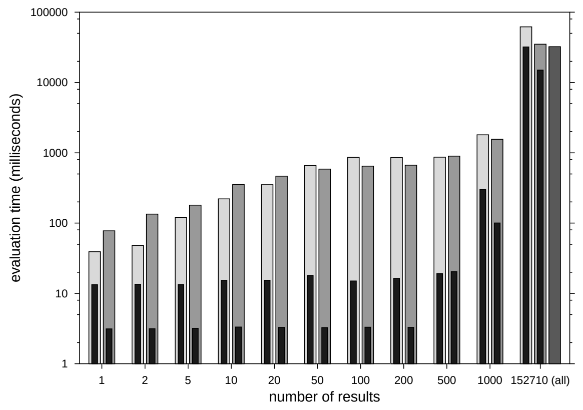
The *struct* query, whose evaluation-time diagram is depicted in Figure 10.2(d), shows a different behavior than the queries with keywords: Here, the *schema* method is the most efficient method, at least if only a prefix of all results is requested. The evaluation time rises only during the retrieval of all results. There is a simple explanation for that behavior: Each element name has far more occurrences than a keyword, and a query that only has structural selectors retrieves many more results than a query with text selectors. In particular, most second-level queries retrieve many results, so that the prefix of n results is retrieved rapidly. The *hybrid* method shows a similar time behavior, but with a larger constant. This meets our expectations because the *schema* method and the *hybrid* method work similarly for structure-only queries. We attribute the constant time difference to the more complex evaluation part of the *hybrid* query processor: The processor always computes the embedding cost of second-level queries (because they *may* have term selectors), and therefore needs larger data structures and more complex algorithms than the *schema* query processor. The *direct* method shows a disadvantageous time behavior, compared to the *schema* method and the *hybrid* method: The low selectivities of structure-only queries produces large lists that store intermediate results. The *direct* method performs many unnecessary list operations, because it cannot determine in advance which combinations of lists will lead to matches.



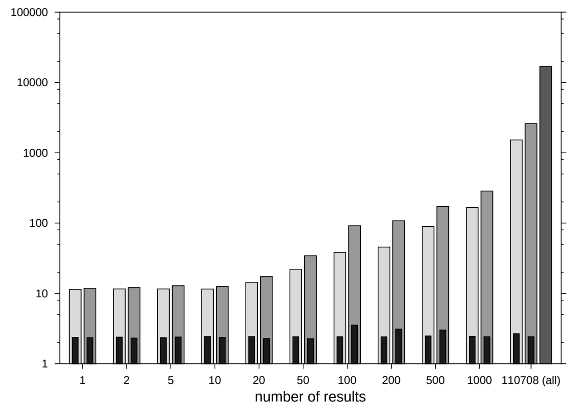
(a) *4x1* query pattern



(b) *3x3* query pattern



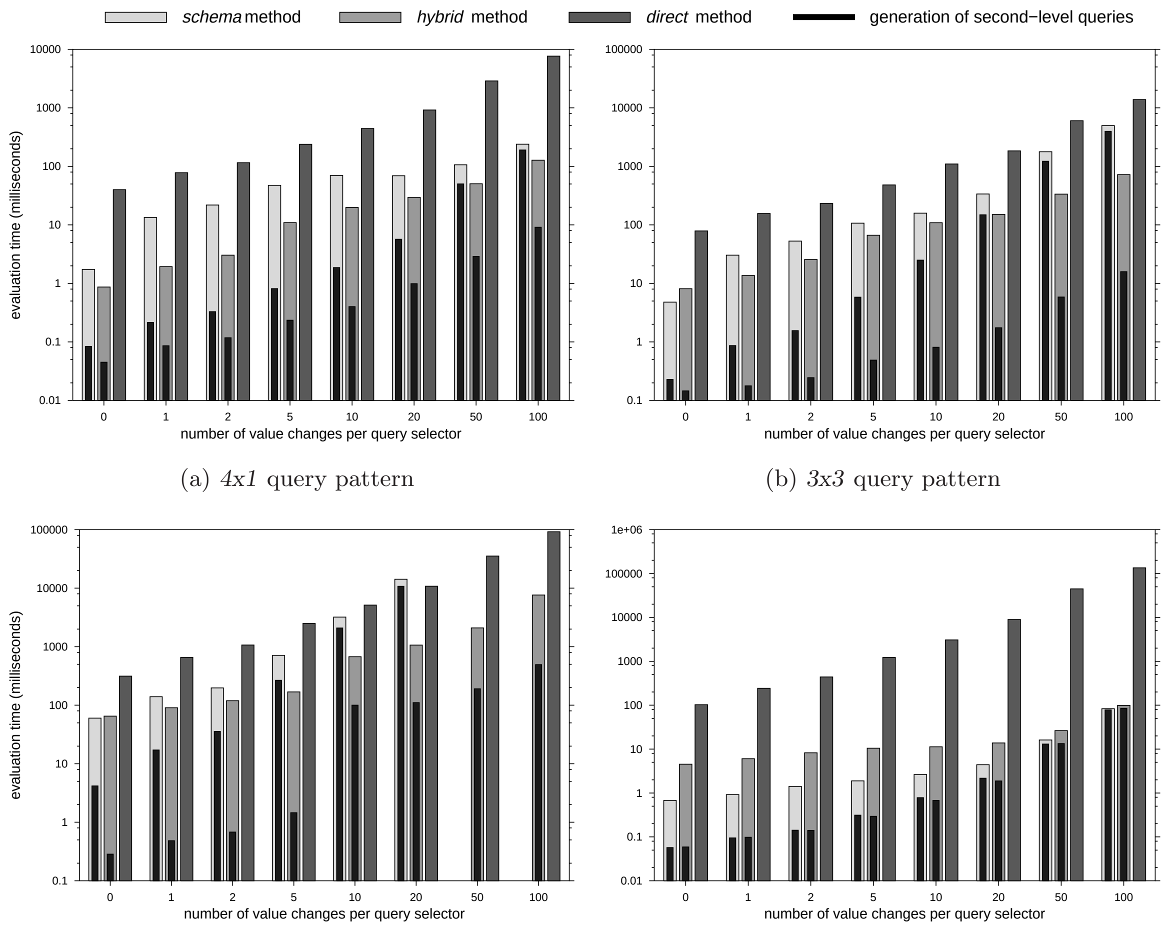
(c) *5x7* query pattern



(d) *struct* query pattern

Figure 10.3: Query-evaluation times for the *dblp* collection depending on the number of requested results. Fixed parameter: 10 value changes per query selector.

Figure 10.3 shows the results obtained for the *dblp* collection. The diagrams resemble those obtained for the *100x100* collection. There is, however, an important difference: The evaluation time of a query using the *schema* method rises less, and the time needed to create the second-level queries takes up a smaller percentage of the total evaluation time. The collections *100x100* and *dblp* differ in their schema sizes (100 versus 41 nodes). The larger the schema of a collection, the more elements with distinct names exist in which a particular term can occur. Because the *schema* method encodes all distinct variants in distinct second-level queries (including all permitted value changes), more second-level queries exist, and each of them selects less results. We investigate that observation in more depth in Sections 10.4.3 and 10.4.4. The *hybrid* method shows similar behavior for both collections.



(a) *4x1* query pattern

(b) *3x3* query pattern

(c) *5x7* query pattern

(d) *struct* query pattern

Figure 10.4: Query-evaluation times for the *100x100* collection depending on the number of value changes per selector. Fixed parameter: 100 requested results.

10.4.2 Hypothesis 2 (Number of Value Changes)

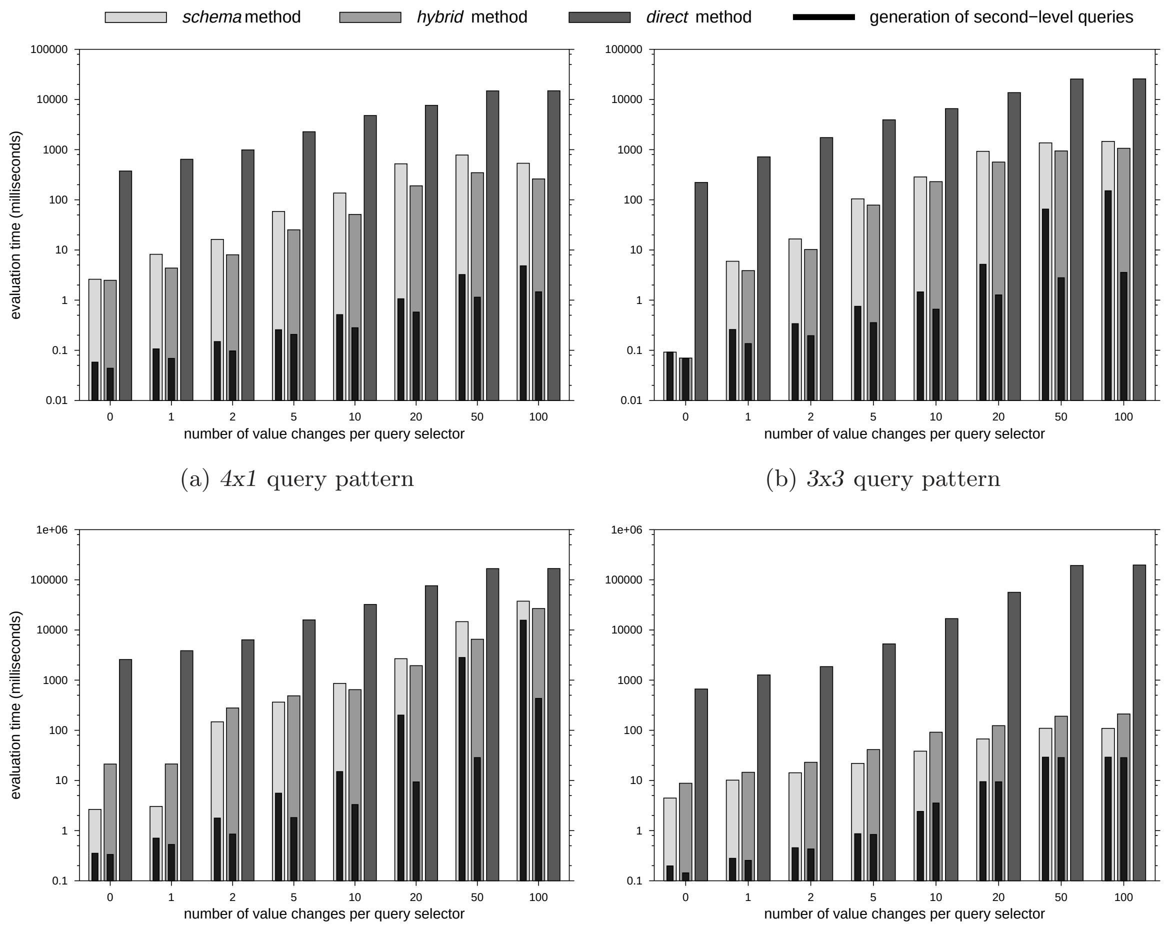
In Hypothesis 2, we assume that the evaluation time of a query increases as the number of value changes per selector increases. Although this seems to be obvious, the hypothesis is not necessarily true because a higher number of value changes also increases the number of results. Thus, it may be possible that the retrieval of the first n results using the *schema* method or the *hybrid* method becomes faster if many value changes are permitted.

The results of our experiments, however, show that the hypothesis is always true. The evaluation time strictly increases as the number of value changes per selector increases. Again, we present a selection of results which were obtained for the collections *100x100* and *dblp*. All diagrams show the evaluation time for a fixed number of 100 requested results.

Figure 10.4 on the page before displays the query evaluation times for the *100x100* collection, requesting 100 results. In the case of the *direct* method, *all* results were evaluated, sorted by increasing cost, and the best 100 results were selected.

The results obtained for the *4x1* query pattern (Figure 10.4(a)) indicate that the *hybrid* method outperforms the other methods in all cases. This is true not only for 100 results, but for any number of results larger than 10. An interesting and typical fact is that the more value changes per selector are permitted, the larger the percentage of time spent for the creation of second-level queries. This is particularly true for the *schema* method, as all four diagrams in Figure 10.4 indicate. If 100 value changes are applied, then the query creation uses up almost all of the evaluation time. During the evaluation of queries belonging to the *5x7* pattern, the *schema* method even exceeded the limit of 100,000 second-level queries if 50 or 100 value changes per selector were applied. Therefore, the evaluation was canceled and the bars for these parameters do not appear in Figure 10.4(c). The *direct* method — although slower than the other methods — has an acceptable response time (circa 100 milliseconds) as long as the number of value changes is low. For many value changes, however, the answer time is unacceptable.

Figure 10.5 on the facing page shows the evaluation times of the queries with respect to the *dblp* collection. Most results are similar to those obtained for the *100x100* collection, but there are two exceptions: First, the bars in all four diagrams and for all three query-evaluation methods grow until 50 value changes are reached. They do not grow any further if 100 value changes are defined. This is caused by the small number of 41 distinct element and attribute names in the *dblp* collection. A number of 50 value changes per selector means that each query selector already matches each element name in the collection. Consequently, the evaluation time of structure-only queries is equal for 50 and 100 value changes, as Figure 10.5(d) shows. The evaluation times of queries generated according to the *4x1* and *3x3* patterns rise only minimally when increasing the number of value changes from 50 to 100. For these queries, only the number of value changes for text selectors rises. These results indicate that the value change of a text selector only has minimal influence on the evaluation time, because the selectivity of a text selector is much higher than that of a structural selector. Second, the time behavior of the *schema* method and the *hybrid* method are very similar. In particular, the *schema* method does not show such a steep ascent as observed with the *100x100* collection. This is obviously a result of the lower amount of time needed for the creation of second-level queries, as the narrow black bars indicate. We conclude that the small schema size, and particularly the very flat hierarchy of the *dblp* collection in conjunction with the higher term selectivities, are beneficial for the *schema* method.



(a) *4x1* query pattern

(b) *3x3* query pattern

(c) *5x7* query pattern

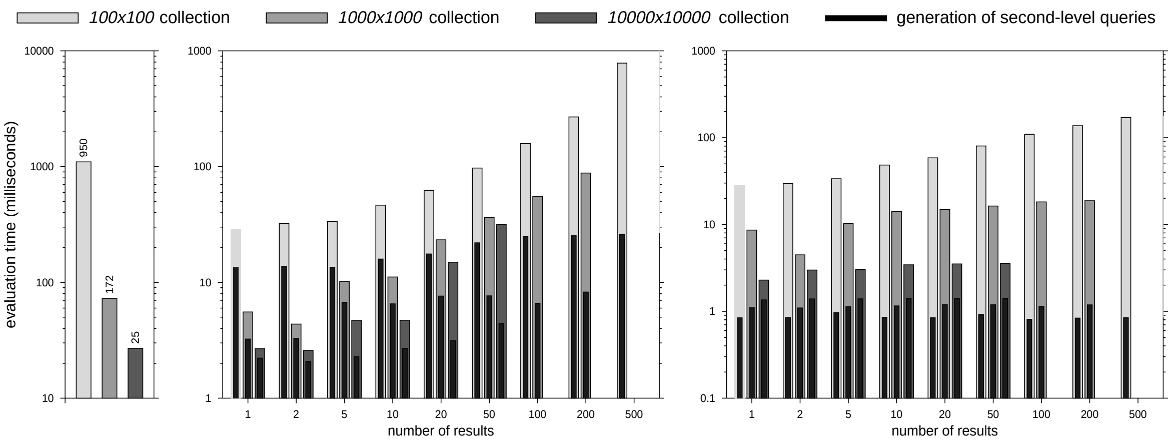
(d) *struct* query pattern

Figure 10.5: Query-evaluation times for the *dblp* collection depending on the number of value changes per selector. Fixed parameter: 100 requested results.

10.4.3 Hypothesis 3 (Schema Size, Varying Number of Names)

In this subsection, we investigate the dependencies of the answer time on the size of the schema. In Hypothesis 3, we assume that the answer times of all three query-evaluation methods decrease as the size of the schema increases, supposed that the number of distinct element names grows according to the schema size. To verify this hypothesis, we use the collections *100x100*, *1000x1000*, and *10000x10000*. Each of these collections was produced using the same parameters for the data generator, except the ones that affect the size of the path tree. These parameters also determine the number of distinct element names, because the data generator creates a unique name for each path-tree node. We only present the results obtained for the *3x3* query pattern.

Figure 10.6 on the next page indicates that our hypothesis holds: The larger the schema, the



(a) *direct* method (b) *schema* method (c) *hybrid* method

Figure 10.6: Evaluation times of the *3x3* query pattern for collections with different schema sizes. For each collection, the number of distinct element names is equal to the number of schema nodes. Fixed parameter: 10 value changes per query selector.

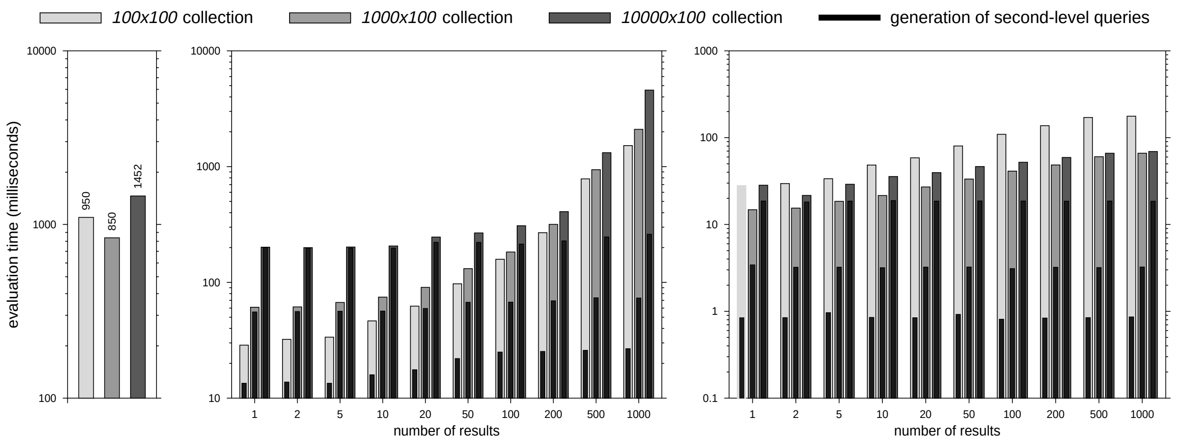
faster the evaluation of the queries. This is particularly true for the *direct* method, which retrieves all existing results and does not make use of the schema. Figure 10.6(a) shows a diagram with three bars representing the evaluation times with respect to the collections *100x100*, *1000x1000*, and *10000x10000*. The number on top of each bar shows the average number of results for 10 queries generated for the *3x3* pattern. The efficiency of the *direct* method depends mainly on the selectivity of the query, and therefore the fewer results that exist, the faster the evaluation.

The efficiency of the *schema* method also becomes better as schema sizes and selectivities are increased. (see Figure 10.6(b)). However, the differences are not as large as observed for the *direct* method, because the large schema size slows down the evaluation time. (We investigate this phenomenon in the next subsection.) The differences between the evaluation times measured for the tree collection even shrink as more results are requested.

The *hybrid* method shows a time behavior superior to that of the *schema* method: With an increasing number of requested results, the evaluation times for large schemata grow even less than the evaluation times for small schemata.

10.4.4 Hypothesis 4 (Schema Size, Constant Number of Names)

In this subsection, we investigate the same question as in the previous one, but with a slightly modified setting: The collections *100x100*, *1000x100*, and *10000x100* that were used in the



(a) *direct* method

(b) *schema* method

(c) *hybrid* method

Figure 10.7: Evaluation times of the *3x3* query pattern for collections with different schema sizes and a constant number of 100 distinct element names. Fixed parameter: 10 value changes per query selector.

experiments have the same sizes as the corresponding collections used in the previous subsection, but all have the same number of 100 distinct element names. Again, we only discuss the results obtained for the *3x3* query pattern.

The first observation concerning Hypothesis 4 is that the evaluation time of queries using the *direct* method is in fact independent of the schema size. It only depends on the total number of results retrieved by a query, as Figure 10.7(a) indicates. Our hypothesis also holds for the *schema* method, as Figure 10.7(b) shows: The larger the schema, the higher evaluation times are. However, the differences between the evaluation times for different schema sizes become less as more results are requested.

The diagram depicted in Figure 10.7(c) indicates that our hypothesis is *not* true for the *hybrid* method: Although the evaluation time of each query type is affected by the schema size, the evaluation times do not always increase as the schema size increases. A closer look at the diagram helps to explain this unexpected behavior: The amount of time spent to *create* the second-level queries strongly depends on the schema size, and therefore our hypothesis holds with respect to query creation. However, the *hybrid* method creates fewer second-level queries than the *schema* method, and the second-level queries have lower selectivities. The more equal-valued nodes that are within the schema (recall that there are only 100 different element names in each collection), the more approximate embeddings of an approXQL query exist, and the more second-level queries must be created in order to retrieve a certain number of results. However, the larger a schema, the shorter the path-dependent index postings belonging to the schema nodes (see Section 7.3). Shorter postings can be processed faster, and therefore the time needed to evaluate a second-level query decreases with increasing schema size. In summary, a collection with a small schema size allows the fast creation of second-level queries, but the evaluation of each of these queries requires considerable time due to long postings. The creation of second-level queries for a collection with a large schema size is expensive, but the evaluation of each second-level query is cheap. Collections with medium-size schemata lead to a good compromise between the fast creation and the fast evaluation of second-level queries. The *schema* method creates far more second-level queries than the *hybrid* method, and therefore has a less balanced relationship between the creation and evaluation of those queries.

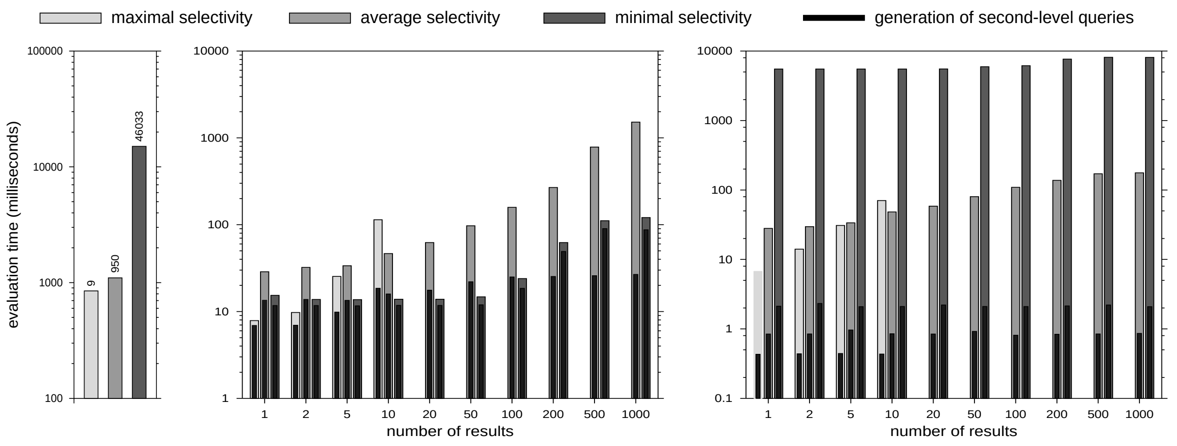
We conclude that there is a strong correlation between the size of a schema and the time needed to create second-level queries. However, only the *schema* method increases the evaluation time of a query with increasing schema size; the *hybrid* method is relatively stable with respect to varying schema sizes.

10.4.5 Hypothesis 5 (Term Selectivity)

In Hypothesis 5, we assume that the evaluation time of a query depends on the selectivities of the query terms. We expect that the *direct* method and the schema-driven methods *schema* and *hybrid* behave differently: If the selectivity decreases, then the *direct* method becomes slower (because it always finds the entire result set), and both the *schema* method and the *hybrid* method become faster (because the first second-level queries find many results).

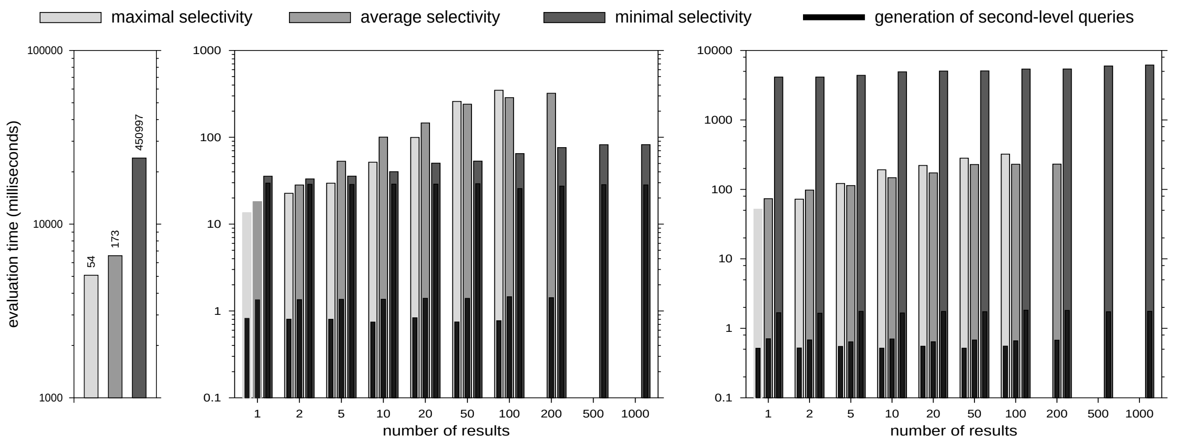
Figure 10.8(a) on the facing page depicts the results obtained for the *3x3* query pattern and the *100x100* collection using the *direct* method. The three bars show the query-evaluation times using maximum, average, and minimum term selectivities. Below the bars, we show the total number of results found for each selectivity mode, where each value is the average of 10 evaluated queries. The diagram indicates that our assumption is true. It is true because the *direct* method must always find all results to retrieve the best n ones. However, a higher term selectivity leads to a large number of results, and a large number of results in turn lead to a high computation time because the lists to process are long. Consequently, a high selectivity leads to slow evaluation times. The diagram obtained for the *dblp* collection (see Figure 10.9(a)) underpins our observation. In fact, the hypothesis holds for all tested query patterns and collections, using any number of value changes.

The *schema* method shows a time behavior that meets our expectations to a great extent (but not fully) as the Figures 10.8(b) and 10.9(b) show: For 1 or 2 requested results, the queries



(a) *direct* method (b) *schema* method (c) *hybrid* method

Figure 10.8: Evaluation times of the *3x3* query pattern for the *100x100* collection with respect to different selectivities of the query terms. Maximum (minimum) term selectivity means that only the least (most) frequent terms of the collection are used; average term frequency means that the terms are selected randomly from the frequency-sorted list of collection terms. Fixed parameter: 10 value changes per query selector.



(a) *direct* method (b) *schema* method (c) *hybrid* method

Figure 10.9: Evaluation times of the *3x3* query pattern for the *dblp* collection with respect to different selectivities of the query terms. Fixed parameter: 10 value changes per query selector.

with maximum term selectivities are the fastest. This means that one of the first second-level queries already found those results. However, we can observe decreasing evaluation times of queries with high selectivities if we request the total number of results (9 for the *100x100* collection and 54 for the *dblp* collection). The minimum-selectivity queries are always faster than their maximum-selectivity and average-selectivity counterparts.

We obtained a different — and unexpected — result for the *hybrid* method (see Figures 10.8(c) and 10.9(c)): The queries with minimum term selectivities have a much higher evaluation time than the queries with average or maximum term selectivities. There is a simple reason for this behavior: The *hybrid* method — in contrast to the *schema* method — selects and merges all postings that belong to a particular query term *or* to one of its alternative values. The lists constructed from the merged postings are long (1,945,725 entries for the most-frequent term in the *100x100* collection, and the 10 most frequent alternative terms), and therefore the computation time increases. Moreover, many of the results selected by the best second-level queries cannot be retrieved immediately, because the subsequent second-level queries may select results with lower costs (see Section 7.5). This also explains why the answer times of minimum-selectivity queries rise less than the answer times of average-selectivity and maximum-selectivity queries.

10.5 Summary

Our experiments show that none of the query-evaluation methods is superior for all queries and collections. In most cases, however, the schema-driven methods are most efficient. They often yield evaluation times that are two or three powers of ten better than those of the *direct* method. When comparing the *schema* method and the *hybrid* method, we observe that there are few cases where the *schema* method is faster than the *hybrid* method. In most cases, however, the *hybrid* method clearly outperforms the *schema* method, benefiting from the merging of second-level queries. The evaluation times of both schema-driven methods may degrade if many second-level queries do not find results. This particularly happens if a query is large, if the selectivities of query terms are high, and if many results are requested. In such cases, the *direct* method is more efficient than the schema-driven methods. Table 10.4 summarizes the dependencies of the query-evaluation methods on selected parameters.

A future version of the `approXQL` query engine should be able to choose the appropriate evaluation method depending on query and data characteristics. For example, the engine could count the number of “`and`” operators in a query and estimate the selectivities of the

criterion	*direct* method	*schema* method	*hybrid* method
increasing result size	$=$	$\uparrow$	$\nearrow$
increasing query size	$\nearrow$	$\uparrow$	$\uparrow$
increasing number of value changes	$\uparrow$	$\uparrow$	$\nearrow$
increasing schema size	$=$	$\nearrow$	$\approx$
increasing term selectivity	$\downarrow$	$\nearrow$	$\downarrow$

Table 10.4: The dependencies of the query-evaluation methods on selected parameters. The symbol "$=$" indicates that a method is by definition independent on a parameter; "$\approx$" indicates a relative independence observed during the tests. The arrows "$\nearrow$" and "$\uparrow$" indicate that the evaluation times (strongly) increase if the parameter value increases; "$\downarrow$" indicates a strong decrease of the evaluation times.

values assigned to the selectors. If both parameters indicate a high query selectivity, the *direct* method should be used. Ideally, the query engine should use a schema-driven method and the *direct* method in parallel. In most cases, the selected schema-driven method will quickly return the best n results. If the evaluation time of this method degrades, then the *direct* method will finally succeed and return all results.